Early Stage Malware Prediction using Machine Learning Techniques and RNN

CHANDAN CHADHA(19BCE1004)

Importing all the required libraries

In [3]:
```
import os
import pandas
import numpy
import pickle
import pefile
import sklearn.ensemble as ek
from sklearn import tree, linear_model
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import SelectFromModel
import joblib
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
from sklearn.pipeline import make_pipeline
from sklearn import preprocessing
from sklearn import svm
from sklearn.linear_model import LinearRegression
```

Loading the initial dataset delimited by |

In [4]:
```
dataset = pandas.read_csv("C:\\Users\\Chandan\\Desktop\\FALL SEM 2021 LABS\\Machine Learning\\J COMPONENT\\ml fir
```

In [3]:
```
dataset.head()
```

Out[3]:

	Name	md5	Machine	SizeOfOptionalHeader	Characteristics	MajorLinkerVersion	MinorLinkerVersion	Size
0	memtest.exe	631ea355665f28d4707448e442fbf5b8	332	224	258	9	0	
1	ose.exe	9d10f99a6712e28f8acd5641e3a7ea6b	332	224	3330	9	0	
2	setup.exe	4d92f518527353c0db88a70fddcfd390	332	224	3330	9	0	
3	DW20.EXE	a41e524f8d45f0074fd07805ff0c9b12	332	224	258	9	0	
4	dwtrig20.exe	c87e561258f2f8650cef999bf643a731	332	224	258	9	0	

5 rows × 57 columns

In [4]:
```
dataset.describe()
```

Out[4]:

	Machine	SizeOfOptionalHeader	Characteristics	MajorLinkerVersion	MinorLinkerVersion	SizeOfCode	SizeOfInitializedData	SizeOfU
count	138047.000000	138047.000000	138047.000000	138047.000000	138047.000000	1.380470e+05	1.380470e+05	
mean	4259.069274	225.845632	4444.145994	8.619774	3.819286	2.425956e+05	4.504867e+05	
std	10880.347245	5.121399	8186.782524	4.088757	11.862675	5.754485e+06	2.101599e+07	
min	332.000000	224.000000	2.000000	0.000000	0.000000	0.000000e+00	0.000000e+00	
25%	332.000000	224.000000	258.000000	8.000000	0.000000	3.020800e+04	2.457600e+04	
50%	332.000000	224.000000	258.000000	9.000000	0.000000	1.136640e+05	2.631680e+05	
75%	332.000000	224.000000	8226.000000	10.000000	0.000000	1.203200e+05	3.850240e+05	
max	34404.000000	352.000000	49551.000000	255.000000	255.000000	1.818587e+09	4.294966e+09	

8 rows × 54 columns

Number of malicious files vs Legitimate files in the training set

In [5]:
```
dataset.groupby(dataset['legitimate']).size()
```

Out[5]:
```
legitimate
```

```
0    96724
1    41323
dtype: int64
```

Dropping columns like Name of the file, MD5 (message digest) and label

In [6]:
```
X = dataset.drop(['Name','md5','legitimate'],axis=1).values
y = dataset['legitimate'].values
```

ExtraTreesClassifier

ExtraTreesClassifier fits a number of randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting

In [7]:
```
extratrees = ek.ExtraTreesClassifier().fit(X,y)
model = SelectFromModel(extratrees, prefit=True)
X_new = model.transform(X)
nbfeatures = X_new.shape[1]
```

ExtraTreesClassifier helps in selecting the required features useful for classifying a file as either Malicious or Legitimate

14 features are identified as required by ExtraTreesClassifier

In [8]:
```
nbfeatures
```

Out[8]: 14

Cross Validation

Cross validation is applied to divide the dataset into random train and test subsets. test_size = 0.2 represent the proportion of the dataset to include in the test split

In [14]:
```
X_train, X_test, y_train, y_test = cross_validation.train_test_split(X_new, y ,test_size=0.2)
```

In [9]:
```
features = []
index = numpy.argsort(extratrees.feature_importances_)[::-1][:nbfeatures]
```

The features identified by ExtraTreesClassifier

In [10]:
```
for f in range(nbfeatures):
    print("%d. feature %s (%f)" % (f + 1, dataset.columns[2+index[f]], extratrees.feature_importances_[index[f]])
    features.append(dataset.columns[2+f])
```

```
1. feature DllCharacteristics (0.141259)
2. feature Characteristics (0.136174)
3. feature Machine (0.102237)
4. feature SectionsMaxEntropy (0.093866)
5. feature MajorSubsystemVersion (0.076185)
6. feature ResourcesMinEntropy (0.054568)
7. feature ResourcesMaxEntropy (0.048843)
8. feature ImageBase (0.047034)
9. feature VersionInformationSize (0.046712)
10. feature SizeOfOptionalHeader (0.041392)
11. feature SectionsMeanEntropy (0.025279)
12. feature Subsystem (0.022657)
13. feature MajorOperatingSystemVersion (0.019587)
14. feature CheckSum (0.019544)
```

Building the below Machine Learning model

In [12]:
```
model = { "DecisionTree":tree.DecisionTreeClassifier(max_depth=10),
          "RandomForest":ek.RandomForestClassifier(n_estimators=50),
          "Adaboost":ek.AdaBoostClassifier(n_estimators=50),
          "GradientBoosting":ek.GradientBoostingClassifier(n_estimators=50),
          "GNB":GaussianNB(),
          "LinearRegression":LinearRegression()
}
```

Training each of the model with the X_train and testing with X_test. The model with best accuracy will be ranked as winner

In [25]:
```
results = {}
for algo in model:
    clf = model[algo]
    clf.fit(X_train,y_train)
    score = clf.score(X_test,y_test)
    print ("%s : %s " %(algo, score))
    results[algo] = score
```

```
RandomForest : 0.994386091996
GradientBoosting : 0.988373777617
GNB : 0.702897500905
DecisionTree : 0.990981528432
LinearRegression : 0.54036008649
Adaboost : 0.986381745744
```

In [26]:
```
winner = max(results, key=results.get)
```

Saving the model

In [27]:
```
joblib.dump(model[winner],'classifier/classifier.pkl')
```

Out[27]:
```
['classifier/classifier.pkl',
 'classifier/classifier.pkl_01.npy',
 'classifier/classifier.pkl_02.npy',
 'classifier/classifier.pkl_03.npy',
 'classifier/classifier.pkl_04.npy',
 'classifier/classifier.pkl_05.npy',
 'classifier/classifier.pkl_06.npy',
 'classifier/classifier.pkl_07.npy',
 'classifier/classifier.pkl_08.npy',
 'classifier/classifier.pkl_09.npy',
 'classifier/classifier.pkl_10.npy',
 'classifier/classifier.pkl_11.npy',
 'classifier/classifier.pkl_12.npy',
 'classifier/classifier.pkl_13.npy',
 'classifier/classifier.pkl_14.npy',
 'classifier/classifier.pkl_15.npy',
 'classifier/classifier.pkl_16.npy',
 'classifier/classifier.pkl_17.npy',
 'classifier/classifier.pkl_18.npy',
 'classifier/classifier.pkl_19.npy',
 'classifier/classifier.pkl_20.npy',
 'classifier/classifier.pkl_21.npy',
 'classifier/classifier.pkl_22.npy',
 'classifier/classifier.pkl_23.npy',
 'classifier/classifier.pkl_24.npy',
 'classifier/classifier.pkl_25.npy',
 'classifier/classifier.pkl_26.npy',
 'classifier/classifier.pkl_27.npy',
 'classifier/classifier.pkl_28.npy',
 'classifier/classifier.pkl_29.npy',
 'classifier/classifier.pkl_30.npy',
 'classifier/classifier.pkl_31.npy',
 'classifier/classifier.pkl_32.npy',
 'classifier/classifier.pkl_33.npy',
 'classifier/classifier.pkl_34.npy',
 'classifier/classifier.pkl_35.npy',
 'classifier/classifier.pkl_36.npy',
 'classifier/classifier.pkl_37.npy',
 'classifier/classifier.pkl_38.npy',
 'classifier/classifier.pkl_39.npy',
 'classifier/classifier.pkl_40.npy',
 'classifier/classifier.pkl_41.npy',
 'classifier/classifier.pkl_42.npy',
 'classifier/classifier.pkl_43.npy',
 'classifier/classifier.pkl_44.npy',
 'classifier/classifier.pkl_45.npy',
 'classifier/classifier.pkl_46.npy',
 'classifier/classifier.pkl_47.npy',
 'classifier/classifier.pkl_48.npy',
 'classifier/classifier.pkl_49.npy',
 'classifier/classifier.pkl_50.npy',
 'classifier/classifier.pkl_51.npy',
 'classifier/classifier.pkl_52.npy',
```

```
'classifier/classifier.pkl_53.npy',
'classifier/classifier.pkl_54.npy',
'classifier/classifier.pkl_55.npy',
'classifier/classifier.pkl_56.npy',
'classifier/classifier.pkl_57.npy',
'classifier/classifier.pkl_58.npy',
'classifier/classifier.pkl_59.npy',
'classifier/classifier.pkl_60.npy',
'classifier/classifier.pkl_61.npy',
'classifier/classifier.pkl_62.npy',
'classifier/classifier.pkl_63.npy',
'classifier/classifier.pkl_64.npy',
'classifier/classifier.pkl_65.npy',
'classifier/classifier.pkl_66.npy',
'classifier/classifier.pkl_67.npy',
'classifier/classifier.pkl_68.npy',
'classifier/classifier.pkl_69.npy',
'classifier/classifier.pkl_70.npy',
'classifier/classifier.pkl_71.npy',
'classifier/classifier.pkl_72.npy',
'classifier/classifier.pkl_73.npy',
'classifier/classifier.pkl_74.npy',
'classifier/classifier.pkl_75.npy',
'classifier/classifier.pkl_76.npy',
'classifier/classifier.pkl_77.npy',
'classifier/classifier.pkl_78.npy',
'classifier/classifier.pkl_79.npy',
'classifier/classifier.pkl_80.npy',
'classifier/classifier.pkl_81.npy',
'classifier/classifier.pkl_82.npy',
'classifier/classifier.pkl_83.npy',
'classifier/classifier.pkl_84.npy',
'classifier/classifier.pkl_85.npy',
'classifier/classifier.pkl_86.npy',
'classifier/classifier.pkl_87.npy',
'classifier/classifier.pkl_88.npy',
'classifier/classifier.pkl_89.npy',
'classifier/classifier.pkl_90.npy',
'classifier/classifier.pkl_91.npy',
'classifier/classifier.pkl_92.npy',
'classifier/classifier.pkl_93.npy',
'classifier/classifier.pkl_94.npy',
'classifier/classifier.pkl_95.npy',
'classifier/classifier.pkl_96.npy',
'classifier/classifier.pkl_97.npy',
'classifier/classifier.pkl_98.npy',
'classifier/classifier.pkl_99.npy',
'classifier/classifier.pkl_100.npy',
'classifier/classifier.pkl_101.npy',
'classifier/classifier.pkl_102.npy',
'classifier/classifier.pkl_103.npy',
'classifier/classifier.pkl_104.npy',
'classifier/classifier.pkl_105.npy',
'classifier/classifier.pkl_106.npy',
'classifier/classifier.pkl_107.npy',
'classifier/classifier.pkl_108.npy',
'classifier/classifier.pkl_109.npy',
'classifier/classifier.pkl_110.npy',
'classifier/classifier.pkl_111.npy',
'classifier/classifier.pkl_112.npy',
'classifier/classifier.pkl_113.npy',
'classifier/classifier.pkl_114.npy',
'classifier/classifier.pkl_115.npy',
'classifier/classifier.pkl_116.npy',
'classifier/classifier.pkl_117.npy',
'classifier/classifier.pkl_118.npy',
'classifier/classifier.pkl_119.npy',
'classifier/classifier.pkl_120.npy',
'classifier/classifier.pkl_121.npy',
'classifier/classifier.pkl_122.npy',
'classifier/classifier.pkl_123.npy',
'classifier/classifier.pkl_124.npy',
'classifier/classifier.pkl_125.npy',
'classifier/classifier.pkl_126.npy',
'classifier/classifier.pkl_127.npy',
'classifier/classifier.pkl_128.npy',
'classifier/classifier.pkl_129.npy',
'classifier/classifier.pkl_130.npy',
'classifier/classifier.pkl_131.npy',
'classifier/classifier.pkl_132.npy',
'classifier/classifier.pkl_133.npy',
'classifier/classifier.pkl_134.npy',
'classifier/classifier.pkl_135.npy',
```

```
 'classifier/classifier.pkl_136.npy',
 'classifier/classifier.pkl_137.npy',
 'classifier/classifier.pkl_138.npy',
 'classifier/classifier.pkl_139.npy',
 'classifier/classifier.pkl_140.npy',
 'classifier/classifier.pkl_141.npy',
 'classifier/classifier.pkl_142.npy',
 'classifier/classifier.pkl_143.npy',
 'classifier/classifier.pkl_144.npy',
 'classifier/classifier.pkl_145.npy',
 'classifier/classifier.pkl_146.npy',
 'classifier/classifier.pkl_147.npy',
 'classifier/classifier.pkl_148.npy',
 'classifier/classifier.pkl_149.npy',
 'classifier/classifier.pkl_150.npy',
 'classifier/classifier.pkl_151.npy',
 'classifier/classifier.pkl_152.npy',
 'classifier/classifier.pkl_153.npy',
 'classifier/classifier.pkl_154.npy',
 'classifier/classifier.pkl_155.npy',
 'classifier/classifier.pkl_156.npy',
 'classifier/classifier.pkl_157.npy',
 'classifier/classifier.pkl_158.npy',
 'classifier/classifier.pkl_159.npy',
 'classifier/classifier.pkl_160.npy',
 'classifier/classifier.pkl_161.npy',
 'classifier/classifier.pkl_162.npy',
 'classifier/classifier.pkl_163.npy',
 'classifier/classifier.pkl_164.npy',
 'classifier/classifier.pkl_165.npy',
 'classifier/classifier.pkl_166.npy',
 'classifier/classifier.pkl_167.npy',
 'classifier/classifier.pkl_168.npy',
 'classifier/classifier.pkl_169.npy',
 'classifier/classifier.pkl_170.npy',
 'classifier/classifier.pkl_171.npy',
 'classifier/classifier.pkl_172.npy',
 'classifier/classifier.pkl_173.npy',
 'classifier/classifier.pkl_174.npy',
 'classifier/classifier.pkl_175.npy',
 'classifier/classifier.pkl_176.npy',
 'classifier/classifier.pkl_177.npy',
 'classifier/classifier.pkl_178.npy',
 'classifier/classifier.pkl_179.npy',
 'classifier/classifier.pkl_180.npy',
 'classifier/classifier.pkl_181.npy',
 'classifier/classifier.pkl_182.npy',
 'classifier/classifier.pkl_183.npy',
 'classifier/classifier.pkl_184.npy',
 'classifier/classifier.pkl_185.npy',
 'classifier/classifier.pkl_186.npy',
 'classifier/classifier.pkl_187.npy',
 'classifier/classifier.pkl_188.npy',
 'classifier/classifier.pkl_189.npy',
 'classifier/classifier.pkl_190.npy',
 'classifier/classifier.pkl_191.npy',
 'classifier/classifier.pkl_192.npy',
 'classifier/classifier.pkl_193.npy',
 'classifier/classifier.pkl_194.npy',
 'classifier/classifier.pkl_195.npy',
 'classifier/classifier.pkl_196.npy',
 'classifier/classifier.pkl_197.npy',
 'classifier/classifier.pkl_198.npy',
 'classifier/classifier.pkl_199.npy',
 'classifier/classifier.pkl_200.npy',
 'classifier/classifier.pkl_201.npy']
```

In [28]:

```
open('classifier/features.pkl', 'w').write(pickle.dumps(features))
```

Calculating the False positive and negative on the dataset

In [41]:

```
clf = model[winner]
res = clf.predict(X_new)
mt = confusion_matrix(y, res)
print("False positive rate : %f %%" % ((mt[0][1] / float(sum(mt[0])))*100))
print('False negative rate : %f %%' % ( (mt[1][0] / float(sum(mt[1])))*100))
```

```
False positive rate : 0.099251 %
False negative rate : 0.147618 %
```

In [36]:

```
# Load classifier
clf = joblib.load('classifier/classifier.pkl')
#load features
features = pickle.loads(open(os.path.join('classifier/features.pkl'),'r').read())
```

Testing with unseen file

Given any unseen test file, it's required to extract the characteristics of the given file.

In order to test the model on an unseen file, it's required to extract the characteristics of the given file. Python's pefile.PE library is used to construct and build the feature vector and a ML model is used to predict the class for the given file based on the already trained model.

In [ ]:

```
# %load malware_test.py
"""
this file extracts the required information of a given file using the library PE

"""

import pefile
import os
import array
import math
import pickle
from sklearn.externals import joblib
import sys
import argparse



def get_entropy(data):
    if len(data) == 0:
        return 0.0
    occurences = array.array('L', [0]*256)
    for x in data:
        occurences[x if isinstance(x, int) else ord(x)] += 1

    entropy = 0
    for x in occurences:
        if x:
            p_x = float(x) / len(data)
            entropy -= p_x*math.log(p_x, 2)

    return entropy


def get_resources(pe):
    """Extract resources :
    [entropy, size]"""
    resources = []
    if hasattr(pe, 'DIRECTORY_ENTRY_RESOURCE'):
        try:
            for resource_type in pe.DIRECTORY_ENTRY_RESOURCE.entries:
                if hasattr(resource_type, 'directory'):
                    for resource_id in resource_type.directory.entries:
                        if hasattr(resource_id, 'directory'):
                            for resource_lang in resource_id.directory.entries:
                                data = pe.get_data(resource_lang.data.struct.OffsetToData, resource_lang.data.str
                                size = resource_lang.data.struct.Size
                                entropy = get_entropy(data)

                                resources.append([entropy, size])
        except Exception as e:
            return resources
    return resources

def get_version_info(pe):
    """Return version infos"""
    res = {}
    for fileinfo in pe.FileInfo:
        if fileinfo.Key == 'StringFileInfo':
            for st in fileinfo.StringTable:
                for entry in st.entries.items():
                    res[entry[0]] = entry[1]
        if fileinfo.Key == 'VarFileInfo':
            for var in fileinfo.Var:
                res[var.entry.items()[0][0]] = var.entry.items()[0][1]
    if hasattr(pe, 'VS_FIXEDFILEINFO'):
          res['flags'] = pe.VS_FIXEDFILEINFO.FileFlags
          res['os'] = pe.VS_FIXEDFILEINFO.FileOS
          res['type'] = pe.VS_FIXEDFILEINFO.FileType
          res['file_version'] = pe.VS_FIXEDFILEINFO.FileVersionLS
          res['product_version'] = pe.VS_FIXEDFILEINFO.ProductVersionLS
```

```
            res['signature'] = pe.VS_FIXEDFILEINFO.Signature
            res['struct_version'] = pe.VS_FIXEDFILEINFO.StrucVersion
    return res

#extract the info for a given file
def extract_infos(fpath):
    res = {}
    pe = pefile.PE(fpath)
    res['Machine'] = pe.FILE_HEADER.Machine
    res['SizeOfOptionalHeader'] = pe.FILE_HEADER.SizeOfOptionalHeader
    res['Characteristics'] = pe.FILE_HEADER.Characteristics
    res['MajorLinkerVersion'] = pe.OPTIONAL_HEADER.MajorLinkerVersion
    res['MinorLinkerVersion'] = pe.OPTIONAL_HEADER.MinorLinkerVersion
    res['SizeOfCode'] = pe.OPTIONAL_HEADER.SizeOfCode
    res['SizeOfInitializedData'] = pe.OPTIONAL_HEADER.SizeOfInitializedData
    res['SizeOfUninitializedData'] = pe.OPTIONAL_HEADER.SizeOfUninitializedData
    res['AddressOfEntryPoint'] = pe.OPTIONAL_HEADER.AddressOfEntryPoint
    res['BaseOfCode'] = pe.OPTIONAL_HEADER.BaseOfCode
    try:
        res['BaseOfData'] = pe.OPTIONAL_HEADER.BaseOfData
    except AttributeError:
        res['BaseOfData'] = 0
    res['ImageBase'] = pe.OPTIONAL_HEADER.ImageBase
    res['SectionAlignment'] = pe.OPTIONAL_HEADER.SectionAlignment
    res['FileAlignment'] = pe.OPTIONAL_HEADER.FileAlignment
    res['MajorOperatingSystemVersion'] = pe.OPTIONAL_HEADER.MajorOperatingSystemVersion
    res['MinorOperatingSystemVersion'] = pe.OPTIONAL_HEADER.MinorOperatingSystemVersion
    res['MajorImageVersion'] = pe.OPTIONAL_HEADER.MajorImageVersion
    res['MinorImageVersion'] = pe.OPTIONAL_HEADER.MinorImageVersion
    res['MajorSubsystemVersion'] = pe.OPTIONAL_HEADER.MajorSubsystemVersion
    res['MinorSubsystemVersion'] = pe.OPTIONAL_HEADER.MinorSubsystemVersion
    res['SizeOfImage'] = pe.OPTIONAL_HEADER.SizeOfImage
    res['SizeOfHeaders'] = pe.OPTIONAL_HEADER.SizeOfHeaders
    res['CheckSum'] = pe.OPTIONAL_HEADER.CheckSum
    res['Subsystem'] = pe.OPTIONAL_HEADER.Subsystem
    res['DllCharacteristics'] = pe.OPTIONAL_HEADER.DllCharacteristics
    res['SizeOfStackReserve'] = pe.OPTIONAL_HEADER.SizeOfStackReserve
    res['SizeOfStackCommit'] = pe.OPTIONAL_HEADER.SizeOfStackCommit
    res['SizeOfHeapReserve'] = pe.OPTIONAL_HEADER.SizeOfHeapReserve
    res['SizeOfHeapCommit'] = pe.OPTIONAL_HEADER.SizeOfHeapCommit
    res['LoaderFlags'] = pe.OPTIONAL_HEADER.LoaderFlags
    res['NumberOfRvaAndSizes'] = pe.OPTIONAL_HEADER.NumberOfRvaAndSizes

    # Sections
    res['SectionsNb'] = len(pe.sections)
    entropy = map(lambda x:x.get_entropy(), pe.sections)
    res['SectionsMeanEntropy'] = sum(entropy)/float(len(entropy))
    res['SectionsMinEntropy'] = min(entropy)
    res['SectionsMaxEntropy'] = max(entropy)
    raw_sizes = map(lambda x:x.SizeOfRawData, pe.sections)
    res['SectionsMeanRawsize'] = sum(raw_sizes)/float(len(raw_sizes))
    res['SectionsMinRawsize'] = min(raw_sizes)
    res['SectionsMaxRawsize'] = max(raw_sizes)
    virtual_sizes = map(lambda x:x.Misc_VirtualSize, pe.sections)
    res['SectionsMeanVirtualsize'] = sum(virtual_sizes)/float(len(virtual_sizes))
    res['SectionsMinVirtualsize'] = min(virtual_sizes)
    res['SectionMaxVirtualsize'] = max(virtual_sizes)

    #Imports
    try:
        res['ImportsNbDLL'] = len(pe.DIRECTORY_ENTRY_IMPORT)
        imports = sum([x.imports for x in pe.DIRECTORY_ENTRY_IMPORT], [])
        res['ImportsNb'] = len(imports)
        res['ImportsNbOrdinal'] = len(filter(lambda x:x.name is None, imports))
    except AttributeError:
        res['ImportsNbDLL'] = 0
        res['ImportsNb'] = 0
        res['ImportsNbOrdinal'] = 0

    #Exports
    try:
        res['ExportNb'] = len(pe.DIRECTORY_ENTRY_EXPORT.symbols)
    except AttributeError:
        # No export
        res['ExportNb'] = 0
    #Resources
    resources= get_resources(pe)
    res['ResourcesNb'] = len(resources)
    if len(resources)> 0:
        entropy = map(lambda x:x[0], resources)
        res['ResourcesMeanEntropy'] = sum(entropy)/float(len(entropy))
        res['ResourcesMinEntropy'] = min(entropy)
        res['ResourcesMaxEntropy'] = max(entropy)
        sizes = map(lambda x:x[1], resources)
        res['ResourcesMeanSize'] = sum(sizes)/float(len(sizes))
        res['ResourcesMinSize'] = min(sizes)
        res['ResourcesMaxSize'] = max(sizes)
    else:
        res['ResourcesNb'] = 0
```

```
        res['ResourcesMeanEntropy'] = 0
        res['ResourcesMinEntropy'] = 0
        res['ResourcesMaxEntropy'] = 0
        res['ResourcesMeanSize'] = 0
        res['ResourcesMinSize'] = 0
        res['ResourcesMaxSize'] = 0

    # Load configuration size
    try:
        res['LoadConfigurationSize'] = pe.DIRECTORY_ENTRY_LOAD_CONFIG.struct.Size
    except AttributeError:
        res['LoadConfigurationSize'] = 0


    # Version configuration size
    try:
        version_infos = get_version_info(pe)
        res['VersionInformationSize'] = len(version_infos.keys())
    except AttributeError:
        res['VersionInformationSize'] = 0
    return res


if __name__ == '__main__':

    clf = joblib.load('classifier/classifier.pkl')
    features = pickle.loads(open(os.path.join('classifier/features.pkl'),'r').read())
    data = extract_infos(sys.argv[1])
    pe_features = map(lambda x:data[x], features)

    res= clf.predict([pe_features])[0]
    print ('The file %s is %s' % (os.path.basename(sys.argv[1]),['malicious', 'legitimate'][res]))
```

Let's run the program to test the file - Skype.exe

In [40]:
```
%run malware_test.py "/home/Chandan/Downloads/Skype.exe"
```

```
The file Skype.exe is legitimate
```

To test for the malicious file, an application has been downloaded from malwr.com

In [38]:
```
%run malware_test.py "/home/Chandan/Downloads/BCN12ui49823.exe"
```

```
The file BCN12ui49823.exe is malicious
```